A Beta Way: A Tutorial For Using Beta Regression in Pychological Research to Analyze Proportional and Percentage Data

Jason Geller[1], Robert Kubinec[2], and Matti Vuorre[3]

[1]Department of Psychology and Neuroscience, Boston College

[2]NYU Abu Dhabi

[3]Tilburg University

Author Note

Jason Geller http://orcid.org/0000-0002-7459-4505

Correspondence concerning this article should be addressed to Jason Geller, Department of Psychology and Neuroscience, Boston College, McGuinn 300, Chestnut Hill, MA 1335, USA, Email: drjasongeller@gmail.com

Abstract

Rates, percentages, and proportional data are widespread in psychology. These data are usually analyzed with methods falling under the general linear model, which are not ideal for this type of data. A better alternative is the beta regession model which is based on the beta distribution. A beta regression can be used to model data that is non-normal,non-linear, heteroscedastic, and bounded between an interval [0,1]. Thus, the beta regression model is well-suited to examine outcomes in psycholgical research expressed as proportions, percentages, or ratios. The overall purpose of this tutorial is to give researchers a hands-on demonstration of how to use beta regression using a real example from the psychological literature. First, we introduce the beta distribution and the beta regression model highlighting crucial components and assumptions. Second, we highlight how to conduct a beta regression in R using an example dataset from the learning and memory literature. Some extensions of the beta model are then discussed (e.g., zero-inflated, zero- one-inflated, and ordered beta). We present accompanying R code throughout. All code to reproduce this paper can be found on Github: link forthcoming

Keywords: Beta regression, tutorial, psychology, learning and memory

A Beta Way: A Tutorial For Using Beta Regression in Pychological Research to Analyze Proportional and Percentage Data

In psychological research, it is common to use response outcomes that are percentages or proportions. For instance, in educational and cognitive research, one popular way to assess learning is by looking at the proportion correct on some test. To illustrate, consider a memory experiment where participants read a short passage on a specific topic. After a brief distractor task, they take a final memory test with 10 short-answer questions, each worth a different number of points (e.g., question 1 might be worth 4 points, while question 2 might be worth 1 point). Your primary interest is in the total number of correct answers out of the total possible points for each question. An important question is how do we analyze this type of data?

On the surface this may appear to be an easy question to answer, but the statistical analysis of proportions can present numerous difficulties that are often not taken into consideration. By definition, proportions are limited to numerical values between, and including, 0 and 1, the variability in the observed proportions usually varies systematically with the mean of the response, and are non-linear. It is quite common, especially in psychology, to analyze proportional outcomes using methods falling under the general linear model (GLM), which include techniques like *t*-tests and ANOVAs. However, there are several issues with the GLM approach. First, the GLM assumes residuals in the model are normally distributed. Second, it assumes an unbounded distribution that can extend from $-\infty$ to ∞. Third, the GLM assumes the outcome is linear. Lastly, the GLM assumes constant residuals—homoscedasticity. These assumptions are often violated when dealing with proportional data, which are typically bounded between 0 and 1, may not follow a normal distribution, and are heteroscedastic in nature [@ferrari2004; @paolino2001]. Adopting a model that does not capture your data accurately can have deleterious consequences, such as missing a true effect when it exists (Type 2 error), or mistaking an effect as real when it is not (Type 1 error). A goal for any researcher trying to draw inferences from their data is to fit a model that accurately captures the important features of the data, and has predictive utility [@yarkoni2017].

The issues related to analyzing proportional data are not new (see (Bartlett, 1936)). Luckily, several analysis strategies are available to deal with them. One approach we highlight here is beta regression (Ferrari & Cribari-Neto, 2004; Paolino, 2001) and some of its alternatives. With the combination of open-source programming languages like R (R Core Team, 2024) and the great community of package developers, it is becoming trivial to run analyses like beta regression. However, adoption of these methods, especially in psychology, is sparse. A quick Web of Science search for a 10 year period spanning 2014-2024 using (TS=(Psychology)) AND TS=(beta regression) as search terms returned fewer than 20 articles. One reason for the lack of adaptation could be the lack of resources available to wider community (but see (Bendixen & Purzycki, 2023; Heiss, 2021; Vuorre, 2019). We attempt to the rectify this herein.

In this article, we plan to (a) give a brief, non-technical overview of the principles underlying beta regression, (b) walk-through an empirical example of applying beta regression using popular frequentist and Bayesian packages in the popular R programming language and (c) highlight the the extensions which are most relevant to researchers in psychology (e.g., zero-inflated, zero-one-inflated, and ordered beta regressions).

To make this tutorial useful, we will use the popular open source programming language R (R Core Team, 2024). We integrate key code chunks into the main text throughout so the reader can follow along. In addition, this manuscript is fully reproducible as it is written with Quarto and is publicly available at this location: link forthcoming.

Because frequentist statistics are still quite popular in psychology we highlight how to perform beta regression using frequentist packages like `betareg` (Cribari-Neto & Zeileis, 2010) and `glmmTMB` [Brooks et al. (2017); useful if you have nested/multilevel data, or working with more complex models)]. We also highlight how to run these models within a Bayesian framework using the `brms` (Bürkner, 2017) package. Our main goal for this tutorial is for it to be maximally useful regardless of statistical proclivities of the user.

Beta distribution

Before we discuss beta regression it is important to build up an intuition about the beta distribution. Going back to our example from the introduction, our main measure (number of correct / number of incorrect) is a continuous outcome varying between 0 and 1. Given this, what kind of distribution can be used to fit this data? The beta distribution is perfect for analyzing outcomes like proportions, percentages, and ratios.[1] The beta distribution has some desirable characteristics that make it ideal for analyzing this type of data: It is continuous, it is limited to numbers that fall between 0 and 1, and it highly flexible—it can take on a number of different distribution shapes. It is important to note that the beta distribution *excludes* numbers that are exactly 0 and exactly 1. That is, it cannot model values that are exactly 0 or 1.

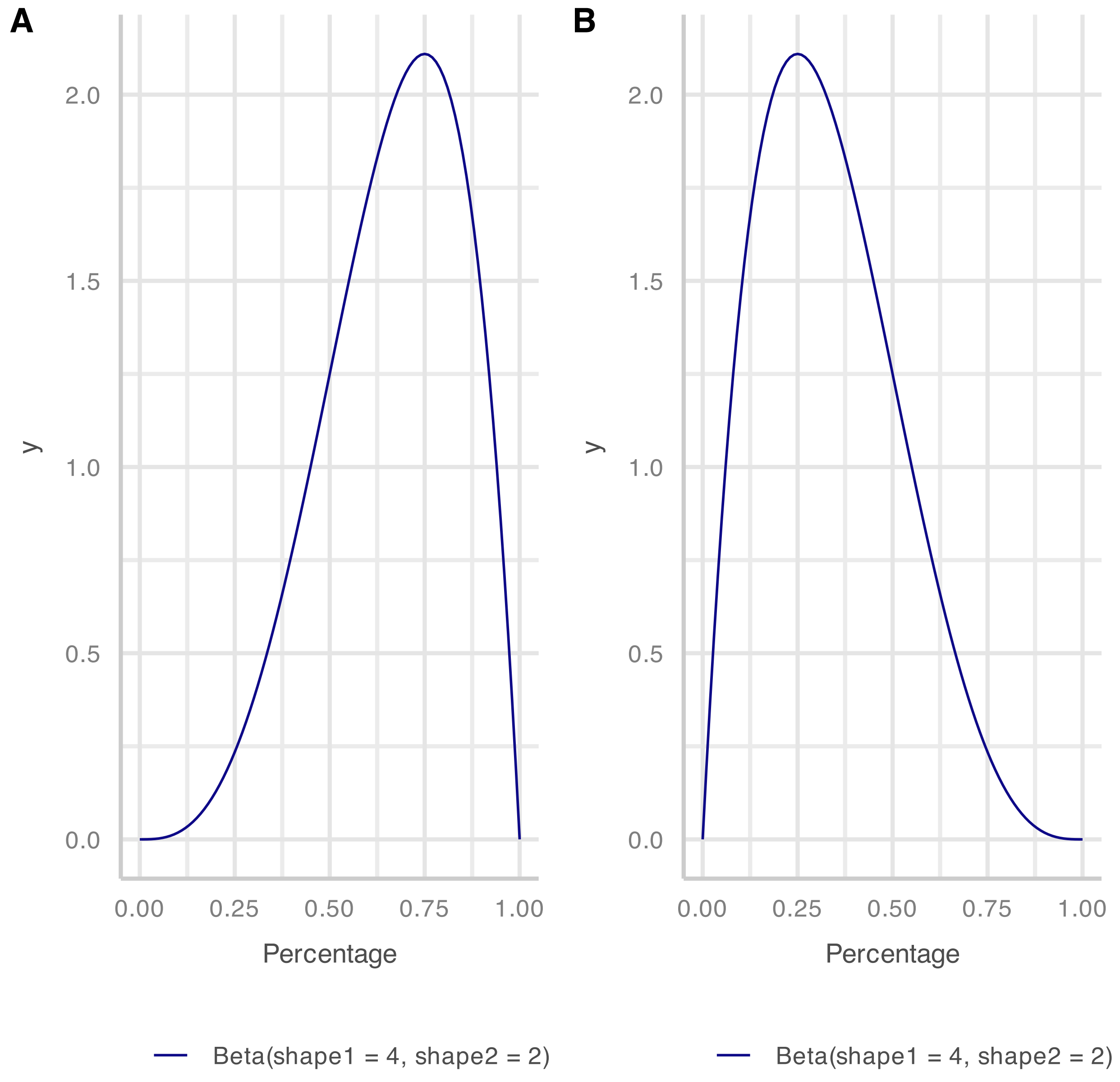
An important freature of the beta distribution is that it can take on a number of different shapes. The location, skew, and spread of the beta distribution is controlled by two parameters: *shape1* and *shape2*. `Shape 1` is sometimes called α and `shape 2` is sometimes called β. Together these two parameters shape the density curve of the distribution. For example, let's suppose a participant got 4 out of 6 correct on a test item. We can take the number of correct on that particular test item (4) and divide that by the number of correct (4) + number of incorrect (2) and plot the resulting density curve. Shape1 in this example would be 4 (number of points received). This parameter reflects the number of successes. `Shape2` would be 2–the number of points not received. This parameter reflects the number of failures. Looking at Figure 1 (a) we see the distribution for one of our questions is shifted towards one indicating higher accuracy on the exam. If we reversed the values of the two parameters Figure 1 (b), we would get a distribution shifted towards 0, indicating a lower accuracy. By adjusting the values of two parameters, we can

[1]Unlike some other popular distributions (Gaussian, poisson, binomial) the beta distribution is not generally thought of as part of the exponential/GLM family.

get a wide range of distributions (e.g., u-shaped, inverted u-shaped , normal, or uniform). As can be seen, the beta distribution is a distribution of proportions or probabilities.

Figure 1

A. Beta distribution with 4 correct and 2 incorrect. B. Beta distribution with 2 correct and 4 incorrect



However, the canonical parametrization of α and β does not lend itself to the regression framework. Thus, it is quite common to talk about μ and ϕ instead, where μ represents the mean or average, and ϕ represents the precision. We can reparameterize α and β into μ and ϕ:

$$[t] \text{ Shape 1: } a = \mu\phi$$

$$\text{Shape 2: } b = (1-\mu)\phi \qquad [t] \text{ Mean: } \mu = \frac{a}{a+b}$$

$$\text{Precision: } \phi = a + b$$

The variance is a function of μ and ϕ:

$$\frac{\mu \cdot (1-\mu)}{1+\phi}$$

Beta regression

We can use regression to model the μ (mean) and ϕ (dispersion) parameters of a beta-distributed response variable. Beta regression is a joint modeling approach that utilizes a logit link function to model the mean of the response variable as a function of the predictor variables. Another link function, commonly the log link, is used to model the dispersion parameter. The application of these links ensures the parameters stay within their respective bounds, with μ between 0 and 1 and ϕ strictly positive. Overall, the beta regression approach respects the bounded nature of the data and allows for heteroskedasticity, making it highly appropriate for data that represents proportions or rates.

Example

Data and Methods

Now that we have built up an intuition about the beta distribution we can start to analyze some data. The principles of beta regression are best understood in the context of a real data set. The example we are gonna use comes from the learning and memory literature. A whole host of literature has shown extrinsic cues like fluency (i.e., how easy something is to process) can influence metamemory (i.e., how well we think we will remember something). As an interesting example, a line of research has focused on instructor fluency and how that influences both metamemory and actual learning. When an instructor uses lots of non-verbal gestures, has variable voice dynamics/intonation, is mobile about the space, and includes appropriate pauses when delivering content, participants perceive them as more fluent, but it does not influence actual memory performance, or what we learn from them [@carpenter2013; @toftness2017; @witherby2022]. While fluency of instructor has not been found to impact actual memory across several studies, @wilford2020 found that it can. In several experiments, @wilford2020 showed that when participants watched multiple videos of a fluent vs. a disfluent instructor (here two videos as opposed to one), they remembered more information on a final test. Given the interesting, and contradictory results, we chose this paper to highlight. In the current tutorial we are going to re-analyze the final recall data from Wilford et al. (2021; Experiment 1a). All their data is open and available here:https://osf.io/6tyn4/.

Accuracy data is widely used in psychology and is well suited for beta regression. Despite this, it is common to treat accuracy data as continuous and unbounded, and analyze the resulting proportions using methods that fall under the general linear model. Below we will reproduce the analysis conducted by Wilford et al. (2020) (Experiment 1a) and then re-analyze it using beta regression. We hope to show how beta regression and its extensions can be a more powerful tool in making inferences about your data.

Wilford et al. (2020) (Expt 1a) presented participants with two short videos highlighting the genetics of calico cats and why skin wrinkles. Participants viewed either disfluent or fluent versions of these videos.[2] For each video, metamemory was assessed using JOLs. JOLs require participants to rate an item on scale between 0-100 with 0 representing the item will not be

[2]See an example of the fluent video here: https://osf.io/hwzuk. See an example of the disfluent video here: https://osf.io/ra7be.

remembered and a 100 representing they will definitely remember the item. In addition, other questions about the instructor were assessed and how much they learned. After a distractor task, a final free recall test was given were participants had to recall as much information about the video as they could in 3 minutes. Participants could score up to 10 points for each video. Here we will only being looking at the final recall data, but you could also analyze the JOL data with a beta regression.

Reanalysis of Wilford et al. Experiment 1a

GLM approach

In Experiment 1a, Wilford et al. (2020) only used the first time point (one video) and compared fluent and disfluent conditions with a *t*-test. In our re-analysis, we will also run a *t*-test, but in a regression context. This allows for easier generalization to the beta regression approach we highlight. Specifically, we will examine accuracy on final test (because the score was on a 10 point scale we multiplied each value by 10 and divided by 100 to get a proportion) as our DV and looking at fluency. Fluency will be dummy coded with the fluent level as our reference level.

Load packages and data. As a first step, we will load the necessary packages along with the data we will be using. While we load all the necessary packages here, we also highlight when packages are needed as code chunks are run.

```
# packages needed
library(tidyverse) # tidy functions/data wrangling/viz
library(betareg) # run beta regression
library(glmmTMB) # zero inflated beta
library(easystats)
library(gghalves)
library(ggbeeswarm)       # Special distribution-shaped point jittering
library(scales) # percentage
```

```r
library(tinytable) # tables

library(marginaleffects) # marginal effects

library(extraDistr)       # Use extra distributions like dprop()

library(brms) # bayesian models


options(scipen = 999) # get rid of scienitifc notation
```

Next, we load in our data, rename the columns to make them more informative, and transform the data. Here we transform accuracy so it is a proportion by multiplying each score by 10 and dividing by 100. Finally, we dummy code the `Fluency` variable (`Flunency_dummy`) setting the fluent condition to 0 and the disfluent condition to 1. A small version of the dataset can be seen in Table 1 along with the data dictionary in Table 2.

```r
# Read the CSV file located in the "MS/data" directory and store it in a

dataframe

fluency_data <- fluency_data <- read.csv(here::here("MS/data/miko_data.csv"))

%>%


  # Rename the columns for better readability

  rename(

    "Participant" = "ResponseID", # Rename "ResponseID" to "Participant"

    "Fluency" = "Condition",      # Rename "Condition" to "Fluency"

    "Time" = "name",              # Rename "name" to "Time"

    "Accuracy" = "value"          # Rename "value" to "Accuracy"

  ) %>%
  # Transform the data
  mutate(
    Accuracy = Accuracy *10 / 100, # Convert Accuracy values to proportions
    Fluency = ifelse(Fluency == 1, "Fluent", "Disflueny"), # rename levels
    Fluency_dummy = ifelse(Fluency == "Fluent", 0, 1),  # Recode
    #Fluency: 1 becomes 0, others become 1
    Fluency_dummy = as.factor(Fluency_dummy) # turn fluency cond to dummy code

  ) %>%

  filter(Time=="FreeCt1AVG") %>% # only choose first time point

  # Drop the column "X" and "time" from the dataframe
  select(-X, -Time)  %>%

  # move columns around

  relocate(Accuracy, .after = last_col())


# Display the first few rows of the modified dataframe
head(fluency_data) %>%
  tt()
```

Table 1

Snippet of dataset

Participant	Fluency	Fluency_dummy	Accuracy
R_00Owxl28JtOADxn	Fluent	0	0.45
R_1DZHq7wW6PhBu7l	Fluent	0	0.30
R_1FfS9t7o3G2waGp	Fluent	0	0.40
R_1gqH4bLsvaqpRRZ	Fluent	0	0.15
R_1i4M7ZdpTcywgbq	Fluent	0	0.50
R_1NyRhBnAB5J2S3r	Fluent	0	0.70

Table 2

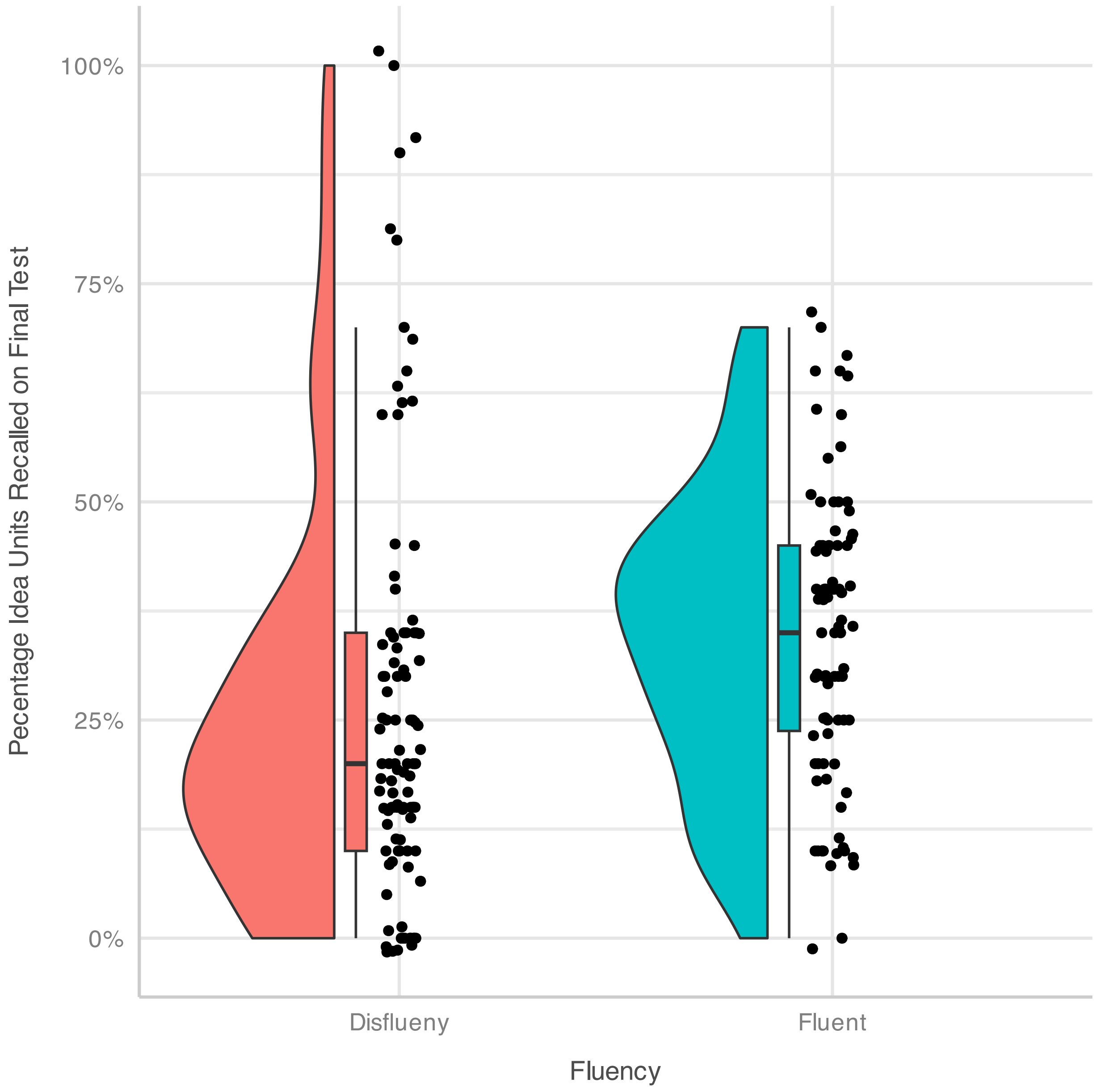
Data dictionary

Column	Key
Participant	Participant ID number
Fluency	Fluent vs. Disfluent
Fluency_dummy	Fluent: 0; Disfluent: 1
Accuracy	Proportion recalled (idea units)

OLS regression. We first start by fitting a regression model using the `lm` function to the data looking at final test accuracy (`Accuracy`) as a function of instructor fluency (`fluency_dummy1`). A quick look at Figure 2 shows us accuracy is higher in the fluent condition vs. the disfluent condition. Is this difference reliable?

Figure 2

Raincloud plot for proprotion recalled on final test as a function of Fluency



Below is the code needed to fit the regression model in R.

```
# fit ols reg

ols_model <- lm(Accuracy~Fluency_dummy, data=fluency_data)
```

```
# for regression model

ols_model_new <- model_parameters(ols_model)
ols_model_new %>%
 tt(digits = 2) %>%
    format_tt(j = "p", fn = scales::label_pvalue()) %>%
  format_tt(escape = TRUE)
```

Table 3

OLS regression model coefficents

Parameter	Coefficient	SE	CI	CI_low	CI_high	t	df_error	p
(Intercept)	0.341	0.031	0.95	0.28	0.40244	11	94	<0.001
Fluency_dummy1	-0.084	0.042	0.95	-0.17	-0.00058	-2	94	0.048

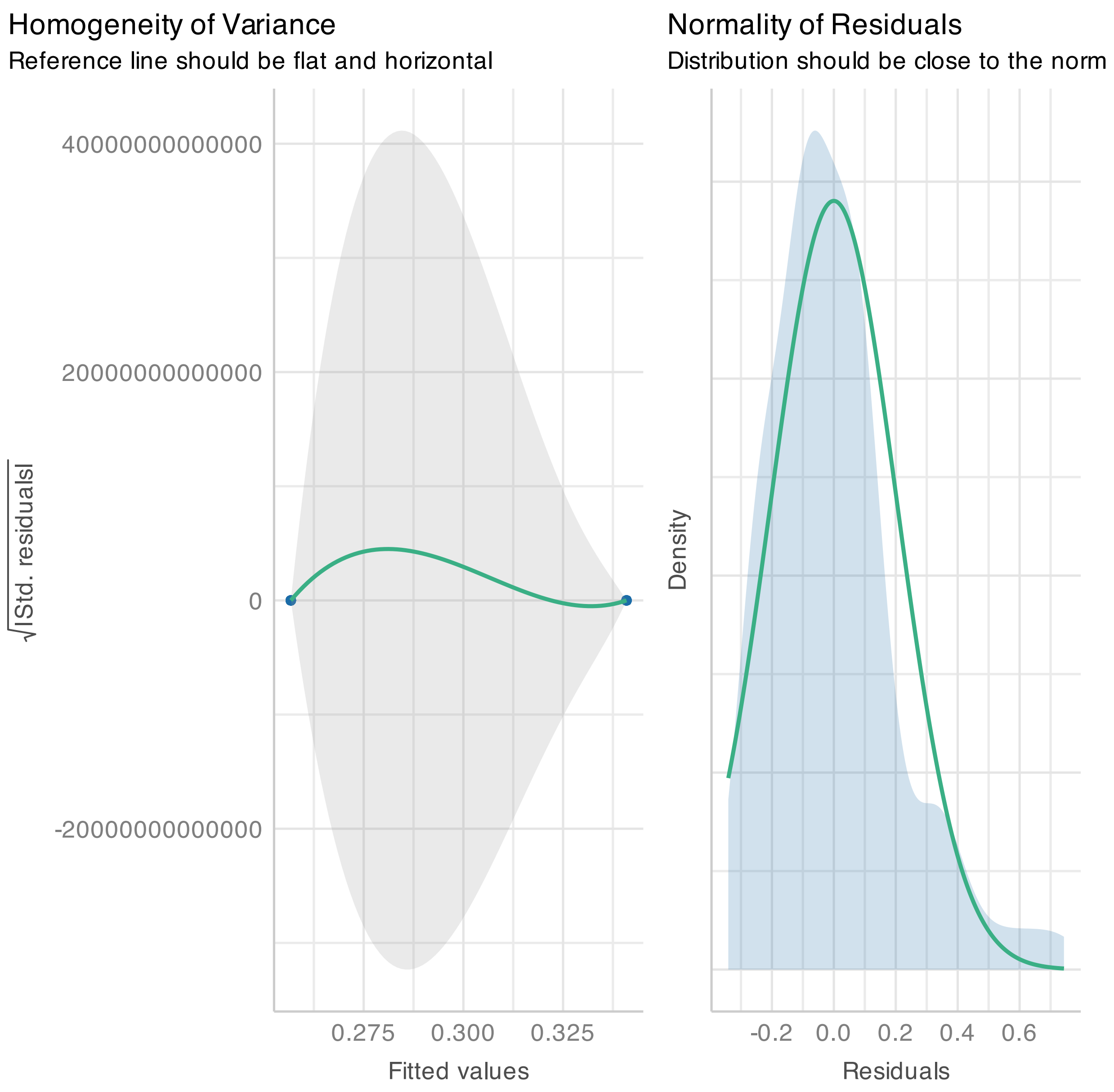
Focusing on output from our regression analysis in Table 3 , we see the that there is a significant effect of Fluency (`Fluency_dummy1`), b = –0.084 , SE = 0.042 , 95% CIs = [–0.168,-0.001], p = 0.048. This is exactly what Wilford et al. (2020) found.

Beta regression approach

Using a traditional approach, we observed instructor fluency impacts actual learning. Keep in mind the traditional approach assumes normality of residuals and homoscadacity. Does the model meet those assumptions? Using `easystats` (Lüdecke et al., 2022) and the `check_model` function, we can easily assess this. In Figure 3 , we see there are some issues with our model. Specifically, there is appears to violations of normality and homoscakdacity.

Figure 3

Assumption checks for OLS model



One solution is to run a beta regression model. Below we fit a beta regression using the `betareg` package (Cribari-Neto & Zeileis, 2010). This a popular package for running frequentist beta regressions.

```
# load betareg package

library(betareg)
```

```r
# running beta model error

beta_model<-betareg(Accuracy~Fluency_dummy, data=fluency_data)
```

```
Error in betareg(Accuracy ~ Fluency_dummy, data = fluency_data): invalid

dependent variable, all observations must be in (0, 1)
```

When you run the above model, an error will appear: `Error in betareg(Accuracy ~ Fluency_dummy, data = fluency_data) : invalid dependent variable, all observations must be in (0, 1)`. If your remember, the beta distribution can model responses in the interval [0-1], but not exactly 0 or 1. We need make sure there are no zeros and ones in our dataset.

Table 4

Number of zeros and ones in our dataset

Accuracy	n
0	9
1	1

Looking at Table 4, we have 9 rows with accuracy of 0, and 1 row with an accuracy of exactly 1. To run a beta regression, we can employ a little hack. We can nudge our 0s towards .01 and our 1s to .99 so they fall within the interval of [0-1].[3]

```r
# transform 0 to 0.1 and 1 to .99

data_beta <- fluency_data %>%

    mutate(Accuracy = ifelse(Accuracy == 0, .01, ifelse(Accuracy == 1, .99,

Accuracy)))
```

Let's fit the model again.

[3]In the newest version of betareg you can model data inclusive 0-1 by including a xdist argumement.

```
# fit beta model without 0s and 1s in our dataset

beta_model<-betareg(Accuracy~Fluency_dummy, data=data_beta)
```

No errors this time! Now, let's interpret the results of our beta regression

Model parameters

μ **component.**

Table 5

Model summary for the mu parameter in beta regression model

Parameter	Coefficient	SE	CI	CI_low	CI_high	z	df_error	p
(Intercept)	-0.59	0.14	0.95	-0.88	-0.31	-4.1	Inf	<0.001
Fluency_dummy1	-0.45	0.2	0.95	-0.83	-0.061	-2.3	Inf	0.023

Looking at the model output in Table 5, the first set of coefficients represents how factors influence the μ parameter, which is the mean of the beta distribution. These coefficients are interpreted on the scale of the logit link function, meaning they represent changes in the log-odds of the mean proportion. The intercept term `(Intercept)` represents the log odds of the mean on accuracy for the fluent instructor condition. Here being in the fluent condition translates to a log odds of –0.593. The fluency coefficient `Fluency_dummy` represents the difference between the fluency and disfluency conditions. That is, watching a fluent instructor leads to higher recall than watching a disfluent instructor, b = –0.447 , SE = 0.197 , 95% CIs = [–0.833,-0.061], p = 0.023.

Predicted probabilities. Parameter estimates are usually difficult to intercept on their own. Instead we should discuss the effects of the predictor on the actual outcome of interest (in this case the 0-1 scale). The logit link allows us to transform back and forth between log-odds and probabilities. By using the inverse of the logit, we can easily transform coefficients to obtain proportions or percentages. In a simple case, we can do this manually, but when we have many moving pieces it can get quite complicated. Thankfully, there is a package called

`marginaleffects` (Arel-Bundock, 2024) that can help us extract the probabilities quite easily.[4] For a more detailed explanation of the package please check out: https://marginaleffects.com/. To get the proportions for each of our categorical predictors we can use the function from the package called `avg_predictions`.

```
#load marginaleffects package

library(marginaleffects)
```

```
# get the predicted probablities for each level of fluency

avg_predictions(beta_model, variables="Fluency_dummy") %>%

  select(-s.value) %>%

  mutate(Fluency_dummy=ifelse(Fluency_dummy==0, "Fluent", "Disfluent")) %>%

  tt(digits = 2) %>%

    format_tt(j = "p", fn = scales::label_pvalue()) %>%

  format_tt(escape = TRUE)
```

Table 6

Predicted probablities for fluency factor

Fluency_dummy	estimate	std.error	statistic	p.value	conf.low	conf.high
Fluent	0.36	0.033	10.8	<0.001	0.29	0.42
Disfluent	0.26	0.027	9.6	<0.001	0.21	0.31

Looking at Table 5, we see that both values in the estimate column are negative, which indicates that probability is below 50%. Looking at the predicted probabilities confirms this. For the `Fluency` factor, we can interpret the estimate column in terms of proportions or percentages. That is, participants who watched the fluent instructor scored on average 36% on the final exam compared to 29% for those who watched the disfluent instructor.

[4]`ggeffects` is another great package to extract marginal effects and plot (Lüdecke, 2018)

Marginal effects. We can also examine marginal effects. Marginal effects are rates-of-change in a given outcome as a function of change in a predictor, holding all else constant in some fashion. Depending on the nature of the question or the nature of the predictor variable, these are typically defined as using either partial derivatives for continuous variables or finite (or first) differences for continuous or categorical variables. There are different types of marginal effects, and various packages calculate them differently. Since we will be using the `marginaleffects` package for this tutorial, we will focus on the average marginal effect (AME), which is used by default in the `marginaleffects` package. AMEs are a good way of summarizing effects because they are most beholden to the original data while also providing a simpler, singular summary of a given effect. In the `marginaleffects` package this involve generating predictions for each row of the original data then averaging these predictions. Using AMEs, one effect size measure we can calculate with categorical variables is the risk difference, which is the discrete difference between the average marginal effect of one condition or group and that of another condition or group. In the `marginaleffects` package, we can use the function `avg_comparisons` to obtain this metric. By default, this function computes the difference. This function can also be used to get other popular effect size metrics, such as odds ratios and risk ratios (see Table 8).

```
# get risk difference

beta_avg_comp<- avg_comparisons(beta_model, comparison= "difference")


beta_avg_comp %>%

  select(-predicted_lo,-predicted_hi,-s.value, -predicted)%>% # remove

unwanted variables

   tt(digits = 2) %>%
    format_tt(j = "p", fn = scales::label_pvalue()) %>%
  format_tt(escape = TRUE)
```

Table 7

Risk difference for fluency factor

term	contrast	estimate	std.error	statistic	p.value	conf.low	conf.high
Fluency_dummy	mean(1) - mean(0)	-0.095	0.042	-2.3	0.023	-0.18	-0.013

Interpreting the output in Table 7, we see the difference between the fluent condition and disfluent condition is 9%. That is, participamnts who watched a fluent instructor scored 9% higher on the final recall test than participants who watched the disfluent instructor, b= –0.095, SE = 0.042, 95 % CIs [–0.177, –0.013], p = 0.023.

We can also get the odds ratio with `avg_comparisons` (see Table 8).

```
# get odds ratios as an example
avg_comparisons(beta_model, comparison = "lnoravg",
    transform = "exp") %>%
  select(-predicted_lo,-predicted_hi,-s.value, -predicted) %>%
  tt() %>%
  format_tt(digits=3)
```

Table 8

Odds ratio for fluency factor

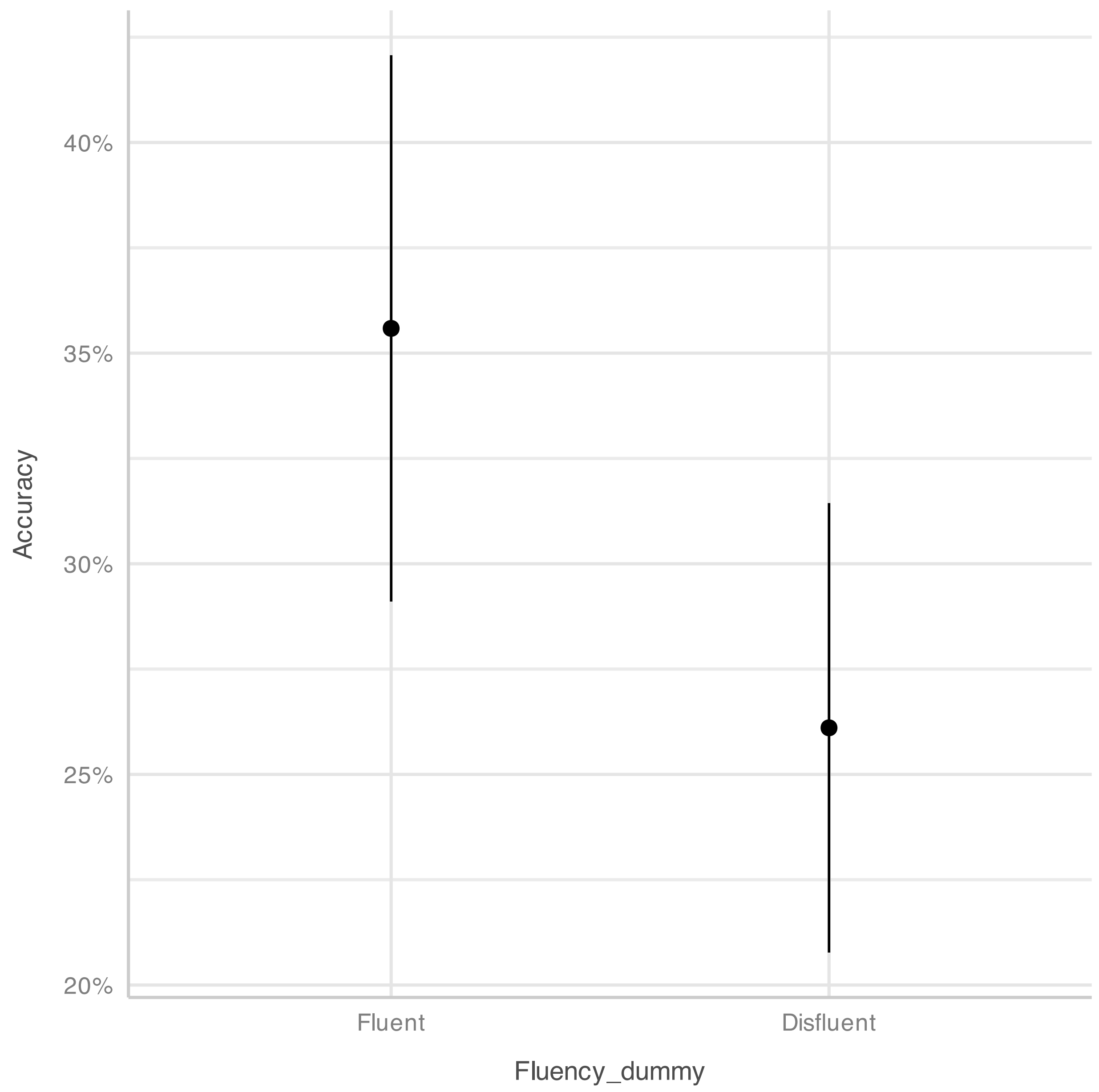
term	contrast	estimate	p.value	conf.low	conf.high
Fluency_dummy	ln(odds(1) / odds(0))	0.639	0.0231	0.435	0.941

Plotting. We can easily plot the μ parameter part of model using the `marginaleffects` package and the `plot_predictions` function (see Figure 4).

```r
plot_predictions(beta_model, condition="Fluency_dummy") +
  theme_lucid(base_size=14) +
  scale_x_discrete(breaks=c("0","1"),
        labels=c("Fluent", "Disfluent")) +
  scale_y_continuous(labels = label_percent())
```

Figure 4

Predicted probablities for fluency factor



Precision (ϕ) component. The other component we need to pay attention to is the dispersion or precision parameter coefficients labeled as `phi` in Table 9 The ϕ parameter tells us how precise our estimate is. Specifically, ϕ in beta regression tells us about the variability of the response variable around its mean. Specifically, a higher dispersion parameter indicates a narrower distribution, reflecting less variability. Conversely, a lower dispersion parameter suggests a wider distribution, reflecting greater variability.

Understanding the dispersion parameter helps us gauge the precision of our predictions and the consistency of the response variable. In `beta_model` we only modeled the dispersion of the intercept. When ϕ is not specified, the intercept is modeled by default.

```r
# fit beta regression model using betareg


beta_model<-betareg(Accuracy~Fluency_dummy, data=data_beta)
```

```r
# get the precision paramter

beta_model %>%

  model_parameters(component = "precision") %>%

   tt(digits = 2) %>%

    format_tt(j = "p", fn = scales::label_pvalue()) %>%

  format_tt(escape = TRUE)
```

Table 9

Beta model summary output of the ϕ parameter

Parameter	Coefficient	SE	CI	CI_low	CI_high	z	df_error	p
(phi)	3.4	0.46	0.95	2.5	4.3	7.5	Inf	<0.001

The intercept under the precision heading is not that interesting. To make things a bit more interesting, let's model the dispersion of the `Fluency` factor—this allows dispersion to differ between the fluent and disfluent conditions. To do this we add a vertical bar to our `betareg` function which allows us to model the dispersion of any factor to the right of it. In the below model, `beta_model_dis`, we model the precision of the `Fluency` factor.

```
# add disp/percison for fluency by including |

beta_model_dis<-betareg(Accuracy~Fluency_dummy | Fluency_dummy,

data=data_beta)
```

```
beta_model_dis  %>%

  model_parameters(component = "precision")%>%

  tt(digits = 2) %>%

    format_tt(j = "p", fn = scales::label_pvalue()) %>%

  format_tt(escape = TRUE)
```

Table 10

Beta regression model summary for fluency factor with ϕ parameter unexponentiated

Parameter	Coefficient	SE	CI	CI_low	CI_high	z	df_error	p
(Intercept)	1.85	0.2	0.95	1.5	2.24	9.2	Inf	<0.001
Fluency_dummy1	-0.94	0.27	0.95	-1.5	-0.41	-3.5	Inf	<0.001

Looking at the precision parameter coefficient for Fluency in Table 10, it is important to note that the estimates are logged and not on the original scale (this is only the case when more than the intercept is modeled). To interpret them on the original scale, we can exponent the log-transformed value—this transformation gets us back to our original scale. We get only the dispersion parameter by setting by setting the `component` argument to `precision` in `model_parameters`. We can also get the original value by including the `exponentiate = TRUE`.

Table 11

Parameter	Coefficient	SE	CI	CI_low	CI_high	z	df_error	p
(Intercept)	6.33	1.27	0.95	4.27	9.38	9.2	Inf	<0.001
Fluency_dummy1	0.39	0.11	0.95	0.23	0.66	-3.5	Inf	<0.001

Beta regression model summary for ϕ parameter exponentiated

In Table 11, The ϕ intercept represents the precision of the fluent condition. The ϕ coefficient for `Fluency_dummy1` represents the change in that precision for the fluent instructors vs. disfluent instructors. The effect is reliable, b = −0.246 , SE = 0.198 , 95% CIs = [−0.634,0.143], p = 0.215.

Now, we have all the parameters to draw two different distributions of our outcome, split by fluency of the instructor. Let's plot these two predicted distributions on top of the true underlying data and see how well they fit. In Figure 5 and Figure 6 and we can see how the distribution changes when we include a dispersion parameter for Fluency.

Note

Horrible fit. Why?

Figure 5

Distributional outcomes of Fluency with disperison parameter included

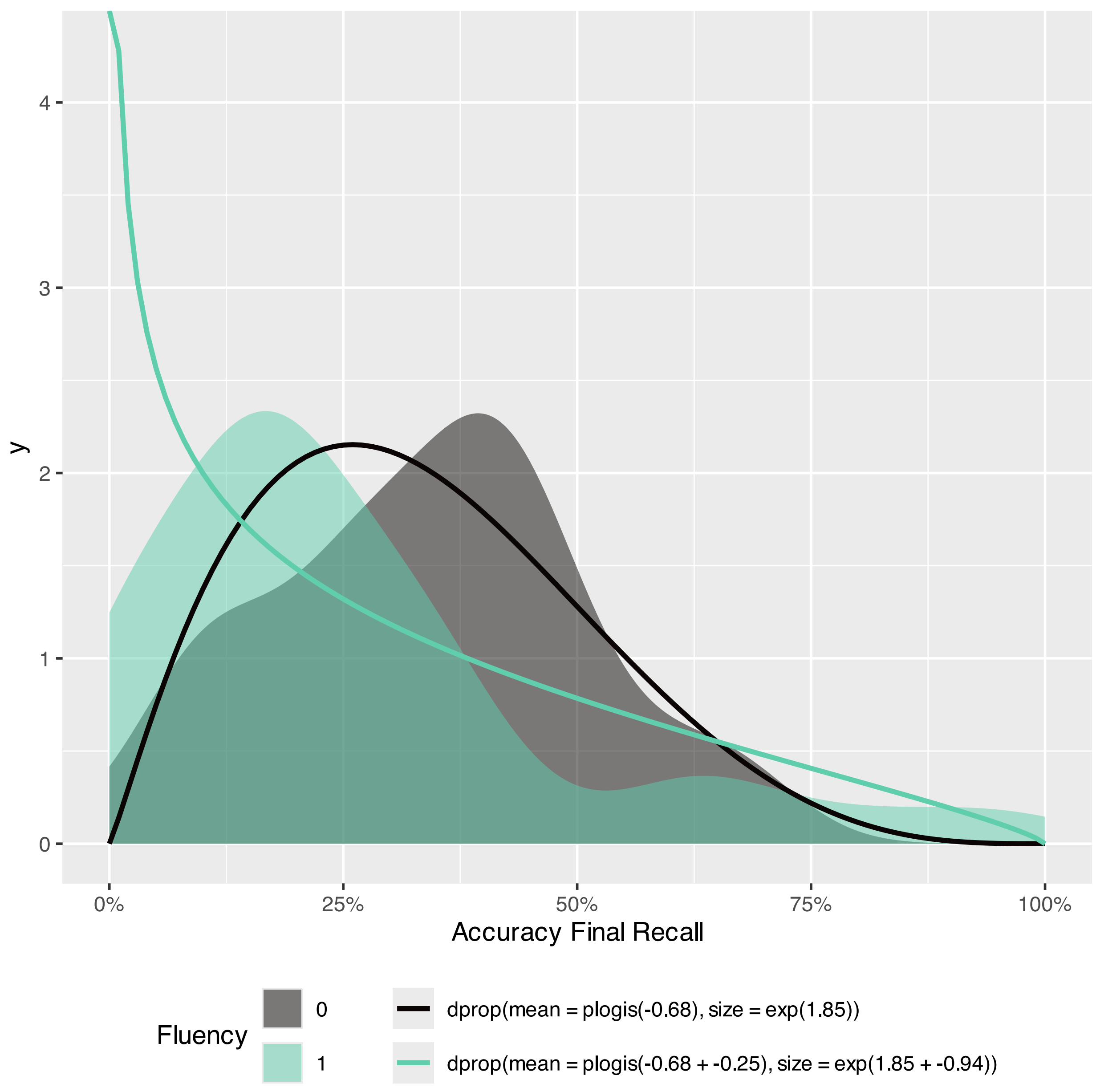
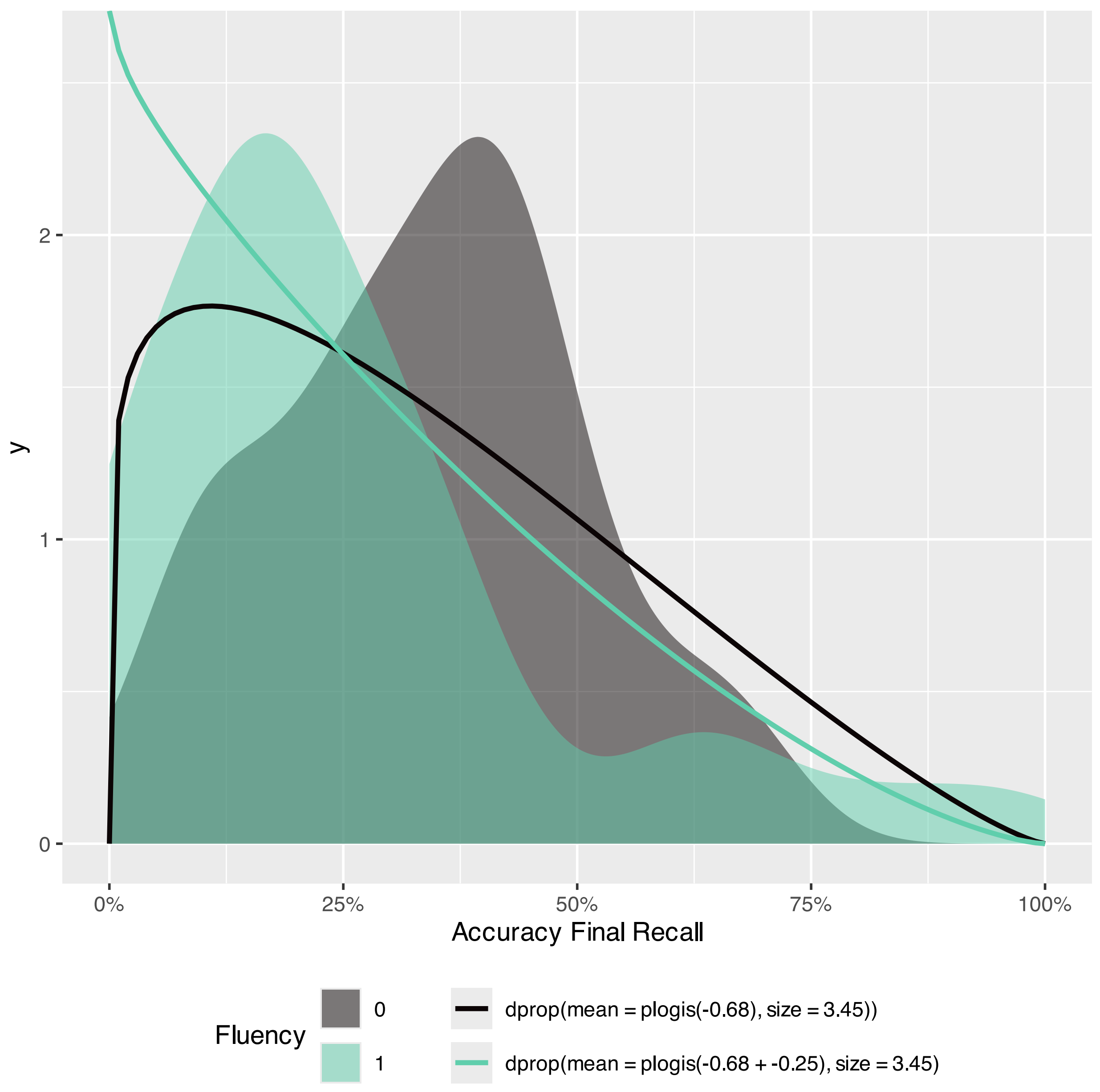


Figure 6

Distributional outcomes of Fluency with no disperison parameter modeled



Note how the whole distribution changes when allow precision to differ across levels of `Fluency`. The data doesn't fit the underlying distribution very well. Despite this, this makes clear the importance of including a precision parameter. A critical assumption of the GLM is homoscedasticity, which means constant variance of the errors. Here we see one of the benefits of a beta regression model. We can include a dispersion parameter for Fluency. Properly accounting for dispersion is crucial because it impacts the precision of our mean estimates and, consequently, the significance of our coefficients. The inclusion of dispersion in the our model changed the statistical significance of the μ coefficient. This suggests that failing to account for the dispersion of the variables might lead to biased estimates. This highlights the potential utility of an approach like beta regression over a traditional GLM (regression or ANOVA approach), as beta regression can explicitly model dispersion and address issues of heteroscedasticity.

We wont always need to include dispersion parameters for each of our variables. We advise conducting a very simple likelihood ratio test (LRT) to examine if we need to include dispersion into our model. To test this we use the `test_likelihoodratio` from the `easystats` ecosystem (Lüdecke et al., 2022).

```
beta_model <- betareg(Accuracy~Fluency_dummy , data=data_beta)


beta_model_dis<-betareg(Accuracy~Fluency_dummy | Fluency_dummy,

data=data_beta)



LRT<-test_likelihoodratio(beta_model, beta_model_dis)


LRT %>%

  tt() %>%

  format_tt(digits=3)
```

Table 12

LRT comparing models with and without a dispersion paramter for fluency

Name	Model	df	df_diff	Chi2	p
beta_model	betareg	3			
beta_model_dis	betareg	4	1	11.5	0.000689

According to the results of our LRT in Table 12 , we would want to model the precision for fluency as the test is significant, $\Delta\chi^2 = 11.5193612$, $p < .001$.

Bayesian implementation of beta regression

We can also fit beta regression models within a Bayesian framework. Adopting a Bayesian framework often provides more flexibility and allows us to quantity uncertainty around our estimates which makes it more powerful than the frequentist alternative. For the purposes of this tutorial, we will not be getting into the minutiae of Bayesian data analysis (i.e., setting informative priors, MCMC sampling, etc,). For a more in-depth look into Bayesian data analysis I refer the reader to XXX.

For the following analyses we will be using default priors provided by `brms`. This will get us something tantamount to a frequentist analysis most of the readers are used to seeing.

To fit our Bayesian models, we will be using a Bayesian package called `brms` (Bürkner, 2017) . `brms` is a powerful and flexible Bayesian regression modeling package that offers built in support for the beta distribution and some of the alternatives we discuss in this tutorial.

We can recreate the beta model from `betareg` in `brms`. Instead of using the `|` operator to specify different parameters like we did in `betareg`, we model each parameter independently. Recall we are fitting two parameters— μ and ϕ. We can easily do this by using the `bf` function from `brms`. `bf()` facilitates the specification of several sub-models within the same formula call. We fit two formulas, one for μ and one for ϕ and store it in `model_beta_bayes`. Here we allow precision to vary as a function of Fluency (this model fit the data better than a model with an intercept-only dispersion parameter).

We first start by loading in the `brms` (Bürkner, 2017) and `cmdstanr` (Gabry et al., 2024)packages.

```
#load brms and cmdstanr

library(brms)
library(cmdstanr) # Use the cmdstanr backend for Stan because it's faster and

more modern than

  # the default rstan You need to install the cmdstanr package first

  # (https://mc-stan.org/cmdstanr/) and then run cmdstanr::install_cmdstan()

to

  # install cmdstan on your computer.

options(marginaleffects_posterior_center = mean)

# get mean instead of median in marginaleffects
```

```
# fit model with mu and phi

model_beta_bayes <- bf(Accuracy  ~ Fluency_dummy, # fit mu model

     phi ~ Fluency_dummy) # fit phi model
```

Using the `bf` function, we specify two models—one for μ and one for ϕ. We then pass `model_beta_bayes` to the `brm` function and set the model family to the beta distribution, which is native to our model using the `brm` function. We also set a bunch of arguments to speed up the fitting of the models, which we will not explain herein.

Table 13

Posterior summary for beta regression using `brms`

Parameter	Mean	CI	CI_low	CI_high	pd	Rhat	ESS
b_Intercept	–0.678	0.95	–0.909	–0.439	1	0.999	4653
b_phi_Intercept	1.808	0.95	1.396	2.172	1	1	3423
b_Fluency_dummy1	–0.247	0.95	–0.66	0.153	0.893	1	3021
b_phi_Fluency_dummy1	–0.924	0.95	–1.439	–0.41	1	0.999	3345

Model parameters

Parameter estimates from the posterior distribution are presented in Table 13.[5] To make the output more readable, each model parameter is labeled with a prefix before the variable name, except for the μ parameter, which takes the same name as the variables in your model. Comparing results with `betareg` in `model_beta`, our results are very similar. Additionally, the parameters can be interpreted in a similar manner and we can use `marginaleffects` to extract marginal effects and risk difference.[6]Contrary to frequentist models, there are no *p*-values to interpret in Bayesian models. There is a metric in the table that is included with models fit when using `easystats` and `bayestestr` called probability of direction (pd) that gives an indication of how much of the posterior distribution estimate is in one direction (positive or negative). The pd measure appears to correlated with *p*-values (see Makowski, Ben-Shachar, & Lüdecke, 2019; Makowski, Ben-Shachar, Chen, et al., 2019). A *pd* of **95%**, **97.5%**, **99.5%** and **99.95%** correspond approximately to two-sided *p*-value of respectively **.1**, **.05**, **.01** and **.001**. Instead of p-values one can look at the 95% credible interval to see if it includes 0–if it does not then the effect can be said to be significant. In the table below the 95% credible intervals are located in the CI_low and CI_high columns. The results are similar to what we found with `betareg` in `model_beta`.

Posterior predictive check

The `pp_check` function allows us to examine the fit between our data and the model. In Figure 7, the x-axis represents the possible range of outcome values and the y-axis represents the density of each outcome value. Ideally, the predictive draws (the light blue lines) should show reasonable resemblance with the observed data (dark blue line). We see it does a pretty good job capturing the data.

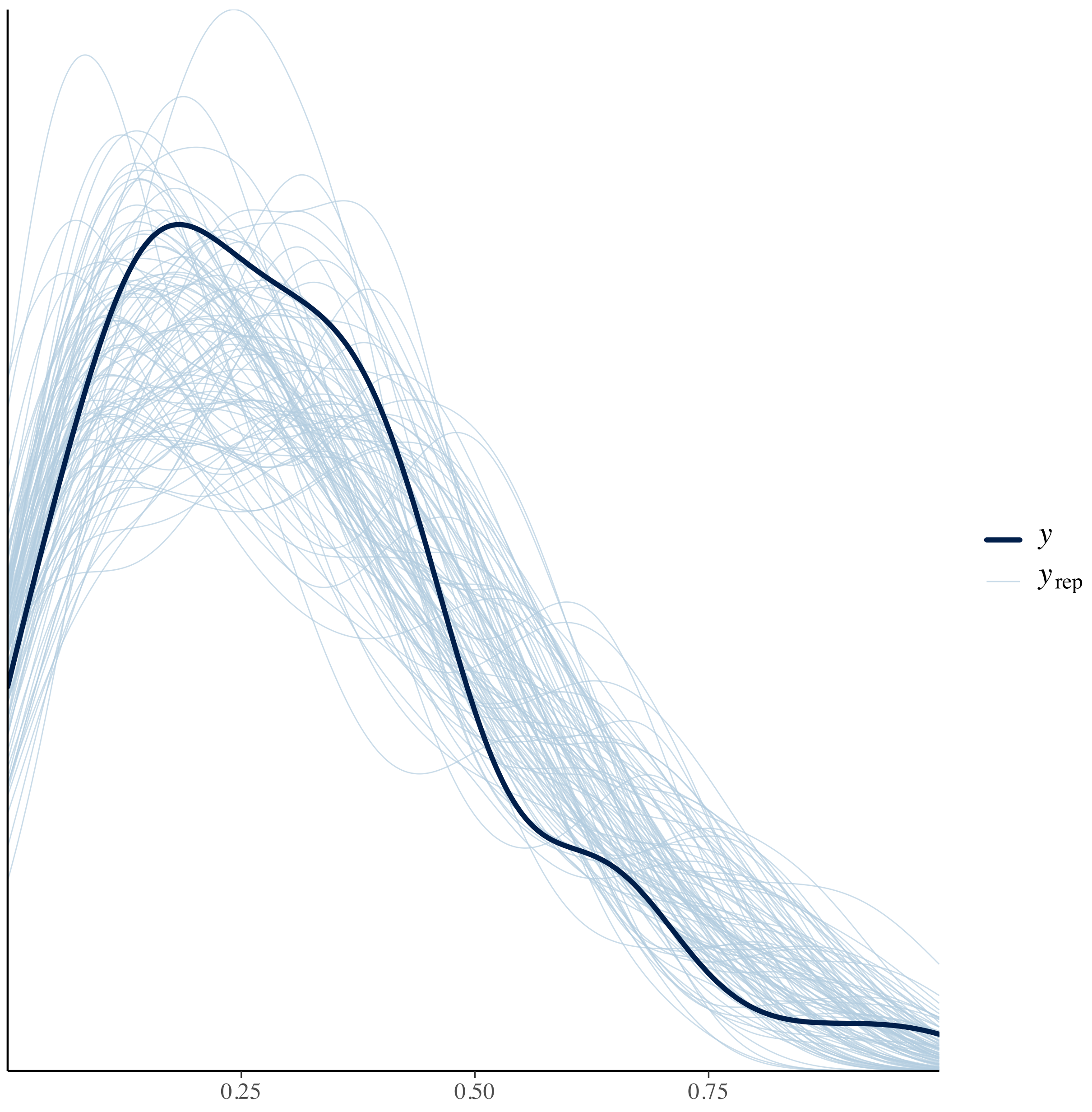
[5]We have chain diagnostics included like Rhat and ESS which indicates how the MCMC sampling performed. For more information check out Gelman et al., 2013; Kruschke, 2014; McElreath, 2020)

[6]It is important to note that these transformations should be applied to draws from the posterior distribution. `Marginaleffects` does this under the hood, but other packages might not.

```r
pp_check(beta_brms,ndraws = 100)
```

Figure 7

Posterior predictive check for our beta model with 100 draws



Predicted probabilities and marginal effects

Predicted probabilities (Table 14) and marginal effects (Table 15) can be computed similarly to our model fit with `betareg`.

```
# get predicted

avg_predictions(beta_brms, variables = "Fluency_dummy") %>%

  tt(digits=3)
```

Table 14

Predicted probablities for fluency factor fit with `brms`

Fluency_dummy	estimate	conf.low	conf.high
0	0.337	0.287	0.392
1	0.285	0.223	0.354

```
# risk difference

beta_brms  %>%

  avg_comparisons(variables = "Fluency_dummy", comparison = "difference") %>%

   select(-predicted_lo,-predicted_hi, -predicted, -tmp_idx) %>%

  tt(digits=3)
```

Table 15

Risk difference for fluency factor fit with `brms`

term	contrast	estimate	conf.low	conf.high
Fluency_dummy	mean(1) - mean(0)	−0.0522	−0.137	0.0337

In Table 15, the risk difference in predicted outcomes is 0.05, which is roughly what we found before with our frequentist model. The 95% credible interval includes zero so we can state the fluency effect is not statistically significant.

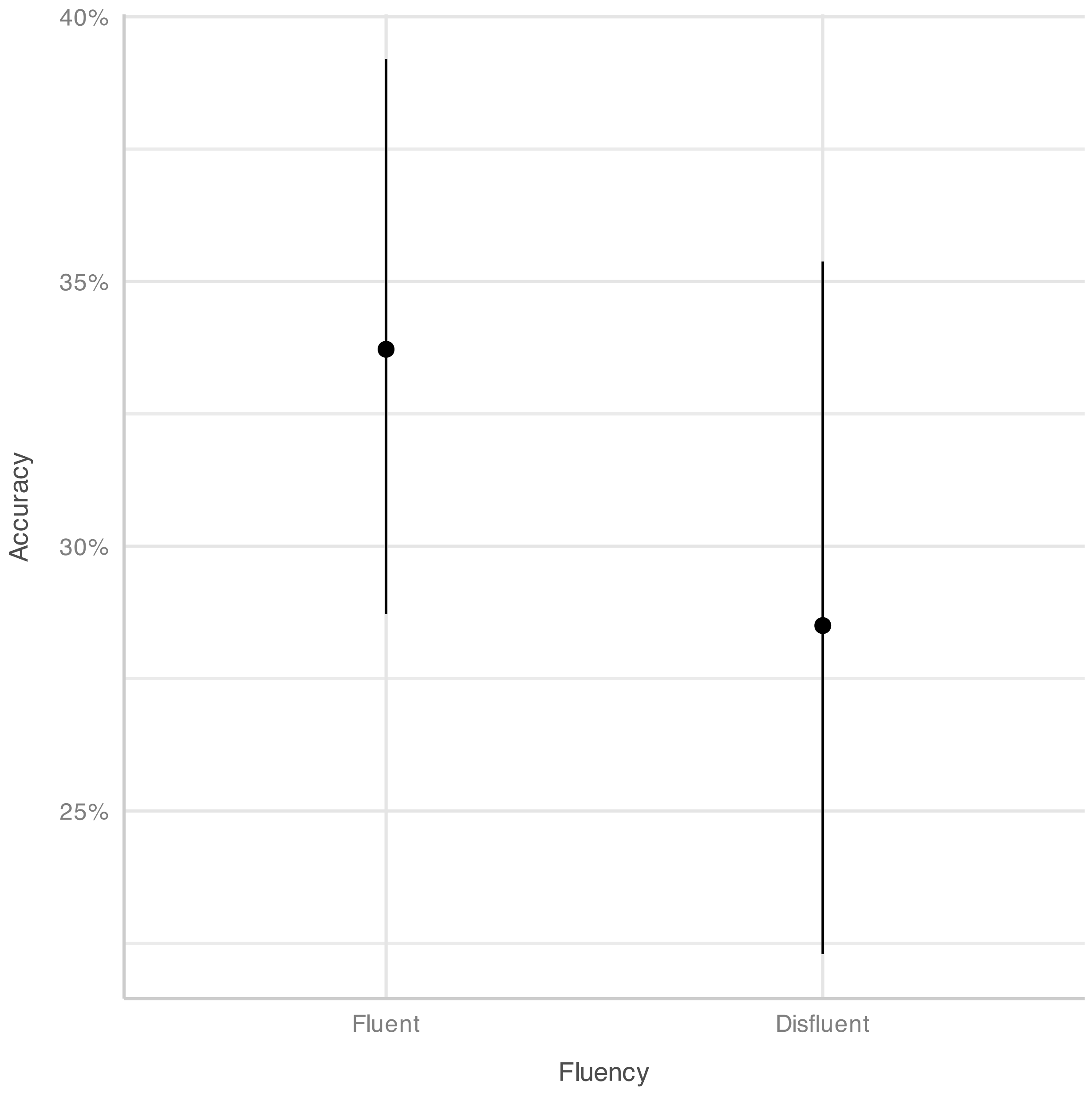
Plotting

Similar to our frequentist model, we can use `plot_predictions` to plot our model on the original scale (see Figure 8)

```
plot_predictions(beta_brms, condition="Fluency_dummy") + theme_lucid(base_size
= 14) +
  scale_x_discrete(breaks=c("0","1"),
        labels=c("Fluent", "Disfluent")) +
scale_y_continuous(labels = label_percent()) +
    labs(x="Fluency")
```

Figure 8

Predicted probablities for fluency with 95% credible intervals



Zero-inflated beta (ZIB) regression

A limitation of the beta regression model is it can can only model values between 0 and 1, but not 0 or 1. In our dataset we have 9 rows with `Accuracy` equal to zero.

To use the beta distribution we nudged our zeros to 0.01–which is never a good idea in practice. In our case it might be important to model this, as fluency of instructor might be an important factor in predicting the zeros in our model. Luckily, there is a model called the zero-inflated beta (ZIB) model that takes into account the structural 0s in our data. We'll still model the $\boxed{?}$ and $\boxed{?}$ (or mean and precision) of the beta distribution, but now we'll also add one new special parameter: $\boxed{?}$. With zero-inflated regression, we're actually modelling a mixture of the data-generating process. The α parameter uses a logistic regression to model whether the data is 0 or not. Below we fit a model called `beta_model_0` using the `glmmTMB` package. The `betareg` model cannot model zero-inflated data. In the `glmmTMB` function, we can model the zero inflation by including an argument called `ziformula`. This allows us to model the new parameter α. Let's fit a model where there is a zero-inflated component for `Fluency_dummy`.

```r
# fit zib modelwith glmmTMB


beta_model_0<-glmmTMB(Accuracy~Fluency_dummy, disp=~Fluency_dummy,  ziformula
= ~ Fluency_dummy, data=data_beta_0, family=beta_family(link="logit"))
```

Model parameters

```r
# use model_parameters to get the summary coefs
model_zi <- model_parameters(beta_model_0)


model_zi %>%
  mutate(p = round(p, 3), p = ifelse(p == 0, "p < .001", p)) %>%
  select(-Effects) %>%
   tt(digits = 2)
```

Table 16

Model summary for ZIB model

Parameter	Coefficient	SE	CI	CI_low	CI_high	z	df_error	p	Component
(Intercept)	−0.631	0.11	0.95	−0.84	−0.42	−5.91	Inf	p < .001	conditional
Fluency_dummy1	−0.031	0.18	0.95	−0.39	0.33	−0.17	Inf	0.863	conditional
(Intercept)	−3.761	1.01	0.95	−5.744	−1.78	−3.72	Inf	p < .001	zero_inflated
Fluency_dummy1	2.056	1.08	0.95	−0.064	4.18	1.9	Inf	0.057	zero_inflated
(Intercept)	2.07	0.2	0.95	1.669	2.47	10.11	Inf	p < .001	dispersion
Fluency_dummy1	−0.85	0.28	0.95	−1.402	−0.3	−3.02	Inf	0.003	dispersion

Table 16 provides a summary of the output for our zib model. As before, the $\boxed{?}$ parameter estimates, which have the conditional tag in the Component column are on the logit scale; while $\boxed{?}$ parameter coefficients (tagged as dispersion in the Component colum) are on the log scale. In addition, the zero-inflated parameter estimates (tagged as zero-inflated in the Component column) are on the logit scale.

Looking at the μ part of the model, there is no significant effect for `Fluency_dummy1`, b = –0.031 , SE = 0.198 , 95% CIs = [–0.634,0.143], p = 0.215 . However, for the zero-inflated part of the model, the `Fluency_dummy1` predictor is significant, b = –0.031 , SE = 0.198 , 95% CIs = [–0.634,0.143], p = 0.215. Lastly the dispersion estimate for `Fluency_dummy1` is significant, b = –3.761 , SE = 0.201 , 95% CIs = [1.453,2.239], p = 0.215.

Predicted probabilities and marginal effects

Similar to above, we can back-transform our estimates to get probabilities. Focusing on the zero-inflated part, we can use the `avg_predictions` function from `marginaleffects` package. Because we are interested in the zero-inflated part of the model we set the `type` argument to `zprob`.

```r
beta_model_0 %>%
  marginaleffects::avg_predictions(by = "Fluency_dummy", type="zprob") %>%
   select(Fluency_dummy, estimate) %>%

   tt(digits = 2)
```

Table 17

Predicted probablites (zero-inflated) for flueny factor

Fluency_dummy	estimate
0	0.023
1	0.154

The estimates provided in Table 17 are percentages of zeros in the model. We can see there are fewer 0s in the fluent condition (2%) compared to the disfluent condition (15%).

We can also get the average marginal effect of Fluency like we did before:

```
  beta_model_0 %>%

   marginaleffects::avg_comparisons(variables = "Fluency_dummy", type="zprob",
 comparison = "difference") %>%

    select(term, contrast, estimate) %>%

   tt()
```

Table 18

Risk difference (zero-inflated) for fluency factor

term	contrast	estimate
Fluency_dummy	mean(1) - mean(0)	0.1311189

Interpreting the estimate in Table 18, seeing lecture videos with a fluent instructor reduces the proportion of zeros by about 13%. Here we have evidence that participants were more likely to do more poorly after watching a disflueny lecture than a fluent lecture.

As a word of caution, the `marginaleffects` package has some issues with fitting models from `glmmTMB` and it has been recommended not to use it. Thus, we will not highlight it here. Please see this issue for more information: https://github.com/vincentarelbundock/marginaleffects/issues/1064

Bayesian implementation of ZIB

Luckily, we can fit a ZIB model using `brms` and use the `marginaleffects` package to make inferences about our parameters of interest. Similar to our beta model we fit in `brms`, we will use the `bf()` function to fit several models. We fit our μ and ϕ parameters as well as our zero-inflated parameter (α; here labeled as `zi`). In `brms` we can use the zero_inflated_beta family argument which is native to `brms`.

```
# fit zero-inflated beta in brms


zib_model <-   bf(
    Accuracy ~ Fluency_dummy,  # The mean of the 0-1 values, or mu
    phi ~ Fluency_dummy,  # The precision of the 0-1 values, or phi
    zi ~ Fluency_dummy,  # The zero-or-one-inflated part, or alpha
  family = zero_inflated_beta()
)
```

Below we pass `zib_model` to the `brm` function.

```
fit_zi <- brm(
  formula = zib_model,
  data = data_beta_0,
  cores = 4,
  iter = 2000,
  warmup = 1000,

  seed = 1234,

  backend = "cmdstanr",

  file = "model_beta_bayes_zib"

)
```

Table 19

Model summary (posterior distribution) for zero-inflated beta model

Parameter	Component	Mean	CI	CI_low	CI_high	pd	Rhat	ESS
b_Intercept	conditional	−0.628	0.95	−0.844	−0.413	1	0.999	5919
b_phi_Intercept	conditional	2.0304	0.95	1.597	2.425	1	1	4743
b_Fluency_dummy1	conditional	−0.0334	0.95	−0.4	0.335	0.57	1	3715
b_phi_Fluency_dummy1	conditional	−0.8382	0.95	−1.399	−0.283	0.998	1	4604
b_zi_Intercept	zero_inflated	−3.8167	0.95	−6.22	−2.224	1	1	1630
b_zi_Fluency_dummy1	zero_inflated	2.1265	0.95	0.278	4.595	0.989	1	1793

Predicted probabilities and marginal effects. We can use `marginaleffects` to get get the predicted probability from our model. To get the zero-inflated part the model, we can set `dpar` argument to `zi` in `avg_predictions`. Table 20 shows predicted probabilities of zero for each level of fluency.

```
fit_zi %>%

 avg_predictions(variables = "Fluency_dummy", dpar="zi") %>%

  tt(digits=3)
```

Table 20

Predicted probablities (zero-inflated) for fluency factor

Fluency_dummy	estimate	conf.low	conf.high
0	0.0315	0.00199	0.0976
1	0.1623	0.07568	0.2722

In Table 21, we get the risk difference between each level of fluency. With a Bayesian implementation we get 95% credible intervals and we see the difference is significant.

```
fit_zi %>%
  avg_comparisons(variables = "Fluency_dummy", dpar="zi", comparison =
"difference") %>%
  tt(digits=3)
```

Table 21

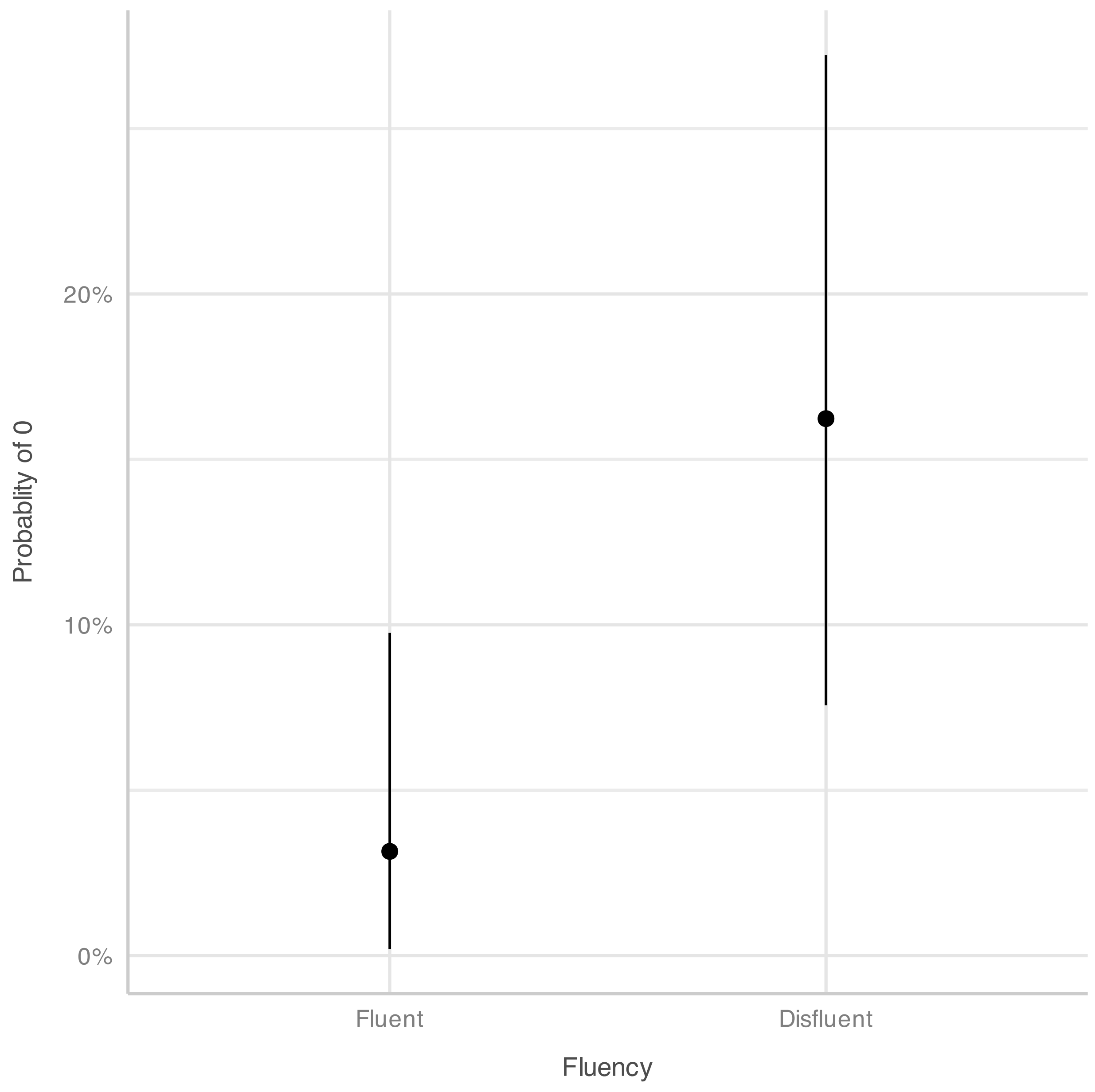
Risk difference (zero-inflated) for fluency factor

term	contrast	estimate	conf.low	conf.high	predicted_lo	predicted_hi	predicted	tmp_idx
Fluency_dummy	mean(1) - mean(0)	0.131	0.0239	0.248	0.025	0.158	0.025	1

Plotting. We can easily plot the zero-inflated part of the model using the `plot_predictions` function (see Figure 9)

Figure 9

Predicted zero-inflated probablities of fluency on final test accuracy



Zero-one inflated beta (ZOIB)

The ZIB model works well if you have 0s in your data, but not 1s. Sometimes it is theoretically useful to model zeros and ones. For example, this is important in visual analog scale data where there might be a prevalence of responses at the bounds (Vuorre, 2019), in JOL tasks (Wilford et al., 2020), or in a free-list task where individuals provide open responses to some question or topic which are then recoded to fall between 0-1 (Bendixen & Purzycki, 2023). Here 0 and 1 are meaningful; 0 means item was not listed and 1 means item was listed first.

In our data, we have exactly one value equal to 1. While probably not significant to alter our findings, we can model ones with a special type of model called the zero-one inflated beta (ZOIB) model. Unfortunately, there is no frequentist implementation of the ZOIB model. Luckily, we can fit a Bayesian implementation of the ZOIB model in `brms`. In this model, we fit four parameters or sub-models. We fit separate models for the mean (μ) and the precision (ϕ) of the beta distribution; a zero-one inflation parameter (i.e. the probability that an observation is either 0 or 1; α); and a 'conditional one inflation' parameter (i.e. the probability that, given an observation is 0 or 1, the observation is 1; γ). This specification captures the entire range of possible values while still being constrained between zero and one.

We use the `bf` function again to fit models for our four parameters. We use the native zero_one_inflated_beta family to fit our model.

```
# fit the zoib model


zoib_model <-   bf(
    Accuracy ~ Fluency_dummy,  # The mean of the 0-1 values, or mu
    phi ~ Fluency_dummy,  # The precision of the 0-1 values, or phi
    zoi ~ Fluency_dummy,  # The zero-or-one-inflated part, or alpha
    coi ~ Fluency_dummy,   # The one-inflated part, conditional on the 0s, or
gamma
  family = zero_one_inflated_beta()
)
```

We then pass the `zoib_model` to our `brm` function. The summary of the output is in Table 22 .

```
# run the zoib mode using brm


fit_zoib <- brm(

  formula = zoib_model,

  data = fluency_data,

  chains = 4, iter = 2000, warmup = 1000,

  cores = 4, seed = 1234,

  backend = "cmdstanr",

  file = "model_beta_zoib_1"

)
```

```
zoib_model <- parameters::model_parameters(fit_zoib, "mean")


zoib_model %>%

 tt(digits=3)
```

Table 22

Model summary (posterior distribution) for the zero-one inflated beta model

Parameter	Mean	CI	CI_low	CI_high	pd	Rhat	ESS
b_Intercept	−0.625	0.95	−0.832	−0.418	1	1	5843
b_phi_Intercept	2.03	0.95	1.609	2.417	1	1	4319
b_zoi_Intercept	−3.818	0.95	−6.16	−2.224	1	1	1782
b_coi_Intercept	−1.856	0.95	−8.827	2.874	0.74	1	2202
b_Fluency_dummy1	−0.202	0.95	−0.539	0.14	0.873	1	3541
b_phi_Fluency_dummy1	−0.431	0.95	−1.02	0.144	0.931	1	4043
b_zoi_Fluency_dummy1	2.27	0.95	0.445	4.696	0.993	1	1930
b_coi_Fluency_dummy1	−0.224	0.95	−5.796	7.326	0.569	1	2348

Model parameters

The output for the model is pretty lengthy—we are estimating four parameters each with their own independent models. All the coefficients are on the logit scale, except ϕ , which is on the log scale. Thankfully drawing inferences for all these different parameters, plotting their distributions, and estimating their average marginal effects looks exactly the same—all the **brms** and **marginaleffects** functions we used work the same.

Note

Should we show how to combine all the submodels and plot the overall effect here?

Plotting ZOIB

Ordered beta regression

Looking at the output from the ZOIB model Table 22, we can see how running a ZOIB model can become vastly complex and computational intensive with larger models as it is fitting submodels for each parameter. A special version of the ZOIB was recently developed called ordered beta regression (Kubinec, 2022). The ordered beta regression model allows for the analysis of continuous data (between 0-1) and discrete outcomes (e.g., 0 or 1). In the simplest sense, the ordered beta regression model is a hybrid model that combines a beta model with ordinal logistic regression model. An in-depth explanation of ordinal regression is beyond the scope of this tutorial (Bürkner & Vuorre, 2019; but see Fullerton & Anderson, 2021). At a basic level, ordinal regression models are useful for outcome variables that are categorical in nature and have some inherent ordering (e.g., Likert scale items). To preserve this ordering, ordinal models rely on the cumulative probability distribution. Within an ordinal regression model, going from one level or category to another is modeled with a single set of covariates that predicts cutpoints between each category. That is, each coefficient shows the effect of moving from one option to a higher option with k–1 cutpoint parameters showing the boundaries or thresholds between the probabilities of these categories. Since there's only one underlying process, there's only one set of coefficients to work with (proportional odds assumption). In an ordered beta regression, three ordered categories are modeled: (1) exactly zero, (2) somewhere between zero and one, and (3) exactly one. In an ordered beta regression, (1) and (2) are modeled with cumulative logits, where one cutpoint is the the boundary between Exactly 0 and Between 0 and 1 and the other cutpoint is the boundary between *Between 0 and 1* and *Exactly 1*. Somewhere between 0-1 (3) is modeled as a beta regression with parameters reflecting the mean response on the logit scale. The ordered beta regression model has shown to be more efficient than some of the methods discussed herein and deserves special mention.

Frequentist implementation

We can run an ordered beta regression using the `glmmTMB` function and changing the family argument to `ordbeta`.

Table 23

Parameter	Coefficient	SE	CI	CI_low	CI_high	z	df_error	p	Component	Effects
(Intercept)	-0.58	0.11	0.95	-0.81	-0.3568	-5.1	Inf	0	conditional	fixed
Fluency_dummy1	-0.31	0.16	0.95	-0.63	0.0031	-1.9	Inf	0.05	conditional	fixed
(Intercept)	6.25		0.95	4.72	8.2866				dispersion	fixed

If we take a look at the summary output in Table 23, we can interpret the values similar to a beta regression, where the conditional effects are on the log odds scale. Here the `Fluency_dummy1` parameter is not statistically significant, $p = .05$.

Predicted probabilities and marginal effects. Remember these values are on the logit scale so we can take the inverse and get predicted probabilities like we have done before. These values are in Table 24.

```
# get the predicted probablities for each level of fluency

avg_predictions(ord_fit, variables="Fluency_dummy") %>%
  select(-s.value)%>%

  tt(digits = 2) %>%

    format_tt(j = "p", fn = scales::label_pvalue()) %>%

  format_tt(escape = TRUE)
```

Table 24

Predicted probablites for flueny factor in ordered beta regression model

Fluency_dummy	estimate	std.error	statistic	p.value	conf.low	conf.high
0	0.36	0.026	14	<0.001	0.31	0.41
1	0.29	0.025	12	<0.001	0.24	0.34

We can get the risk difference as well. These values are in Table 25.

```
# get risk difference

avg_comparisons(ord_fit,variables="Fluency_dummy",  comparison= "difference")

%>%

      select(-predicted_lo,-predicted_hi,-s.value, -predicted) %>%

tt(digits = 2) %>%

    format_tt(j = "p", fn = scales::label_pvalue()) %>%

  format_tt(escape = TRUE)
```

Table 25

Risk difference for fluency factor in ordered beta model

term	contrast	estimate	std.error	statistic	p.value	conf.low	conf.high
Fluency_dummy	mean(1) - mean(0)	-0.069	0.035	-1.9	0.052	-0.14	0.00045

Bayesian implementation

To fit an ordered beta regression in a Bayesian context you can use `ordbetareg` (Kubinec, 2023) package.

We first load in the `ordbetareg` package.

```
# load ordbetareg package

library(ordbetareg)
```

The `ordbetareg` package uses `brms` on the front-end and is straightforward to run. Instead of the `brm` function we use `ordbetareg`. It has similar arguments.

```
# use ordbetareg to fit model

ord_fit_brms <- ordbetareg(Accuracy ~ Fluency_dummy,

                      data=fluency_data,

                      chains=4,

                      iter=2000,

                      backend="cmdstanr",

                      file = "model_beta_ordbeta")
```

Model parameters

Table 26 presents the model summary for our model.

```
ord_fit_brms %>%

  model_parameters() %>%

  tt() %>%

  format_tt(digits=3)
```

Table 26

Model summary for ordered beta model

Parameter	Component	Median	CI	CI_low	CI_high	pd	Rhat	ESS
b_Intercept	conditional	−0.579	0.95	−0.814	−0.3534	1	1	4107
b_Fluency_dummy1	conditional	−0.31	0.95	−0.625	0.0113	0.971	1	3996
phi	distributional	6.177	0.95	4.6	8.1189	1	1	4267

Ordered beta model fit

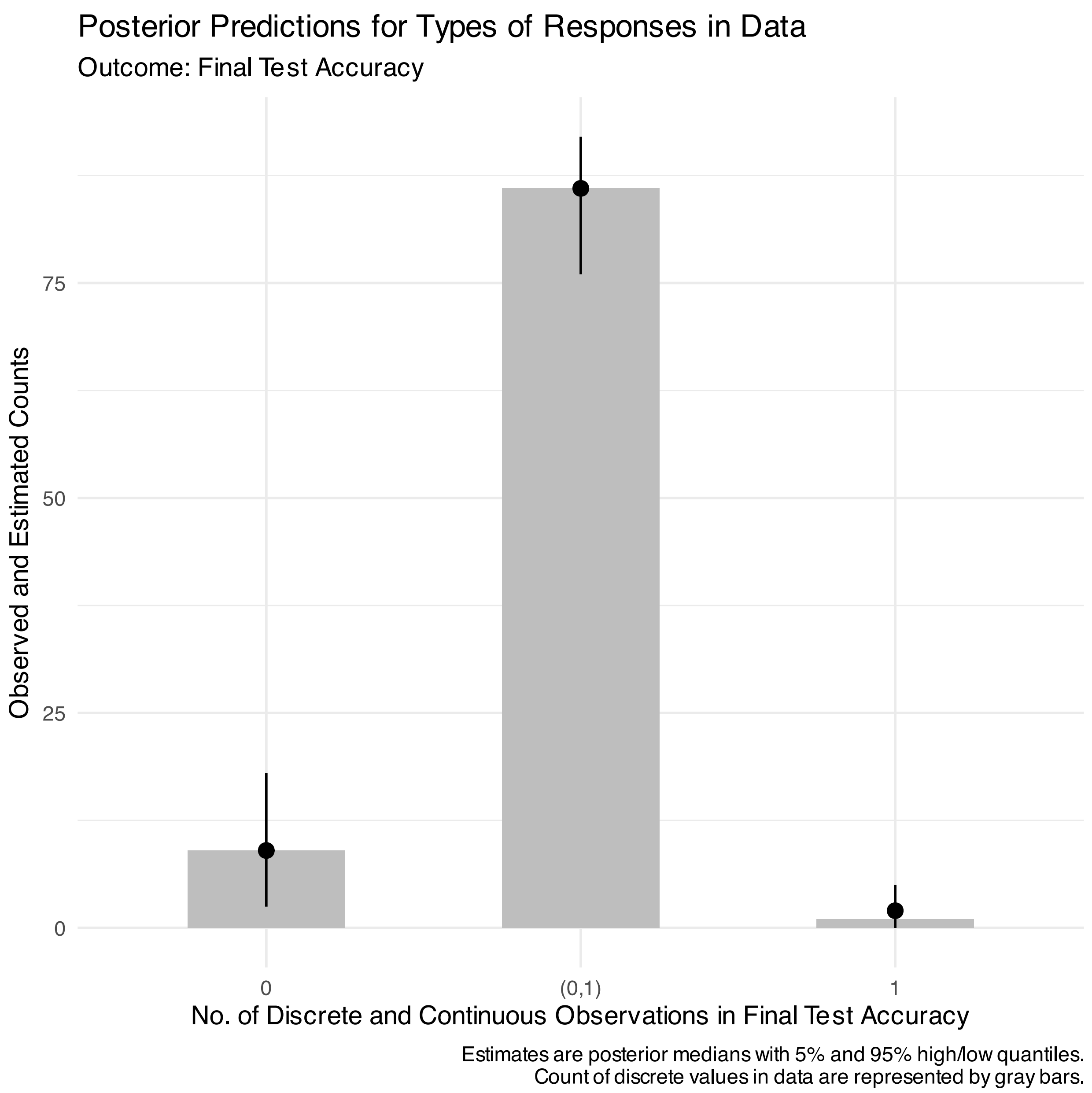
The best way to visualize model fit is to plot the full predictive distribution relative to the original outcome. Because ordered beta regression is a mixed discrete/continuous model, a separate plotting function, `pp_check_ordbetareg`, is included in the `ordbetareg` package that accurately handles the unique features of this distribution. This function returns a list with two plots, `discrete` and `continuous`, which can either be printed and plotted or further modified as `ggplot2` objects. This can be observed in fig x

ℹ Note

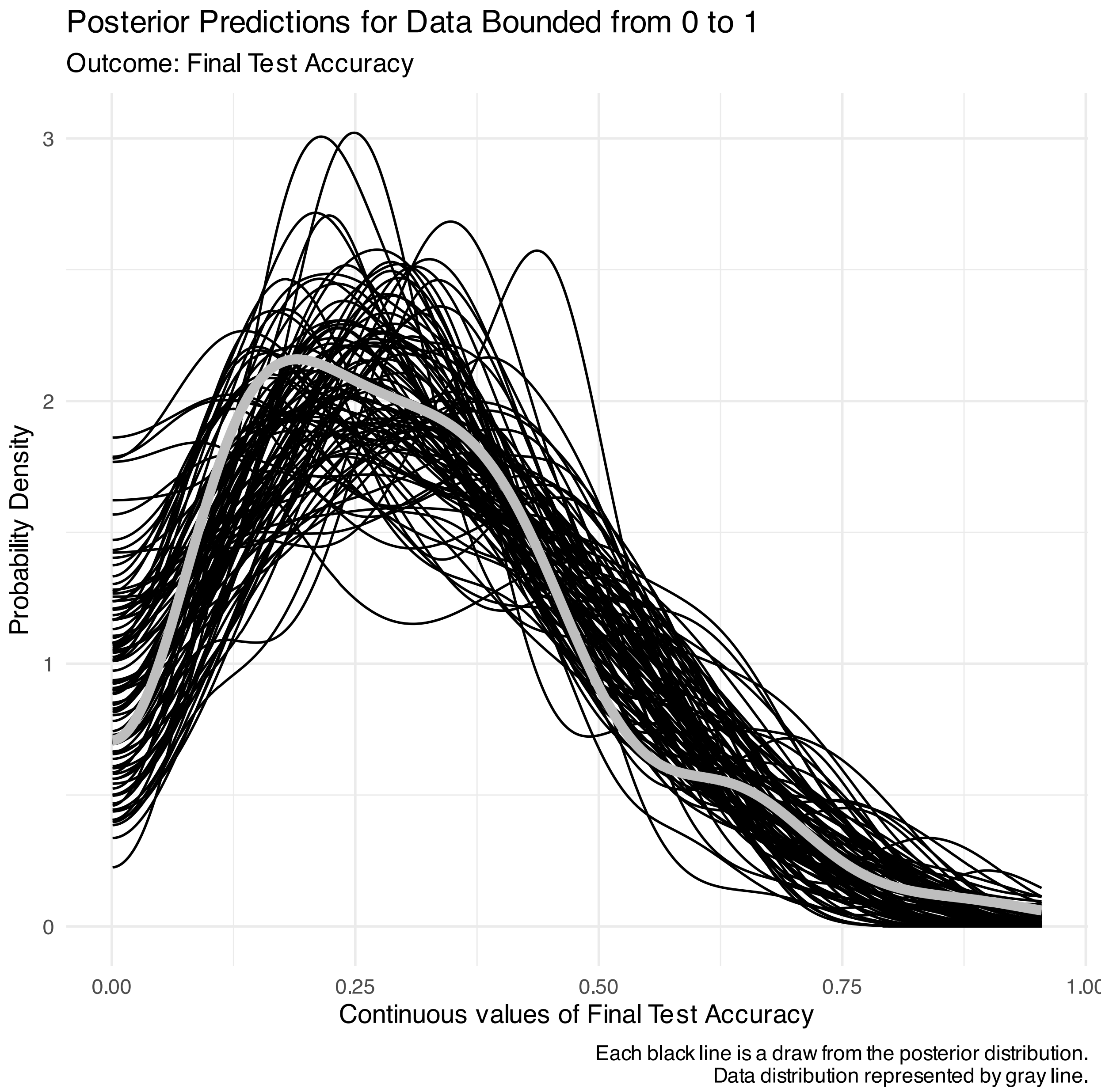
Some reason cant get pdf to render if this figure is labeled

```
plots <- pp_check_ordbeta(ord_fit_brms,
                          ndraws=100,
                          outcome_label="Final Test Accuracy")


plots
```

```
$discrete
```


Posterior Predictions for Types of Responses in Data
Outcome: Final Test Accuracy
Observed and Estimated Counts
0
25
50
75
0
(0,1)
1
No. of Discrete and Continuous Observations in Final Test Accuracy
Estimates are posterior medians with 5% and 95% high/low quantiles.
Count of discrete values in data are represented by gray bars.

```


$continuous
```



The discrete plot which is a bar graph, shows that the posterior distribution accurately captures the number of different types of responses (discrete or continuous) in the data.

For the continuous plot shown as a density plot with one line per posterior draw, the model does a very good job at capturing the distribution.

Overall, it is clear from the posterior distribution plot that the ordered beta model fits the data well.

Discussion and Conclusion

The use of beta regression in psychology, and the social sciences in general, is rare. With this tutorial, we hope to change this. Beta regression models are an attractive alternative to models subsumed under the GLM, which imposes the unrealistic assumptions of normality, linearity, homoscedasticity, and requires the data to be unbounded. Beyond the GLM, there is a diverse array of different models that can be used depending on your outcome of interest.

Throughout this tutorial our main aim was to help guide researchers in running analyses with proportional or percentage outcomes using beta regression and some of it's alternatives. In the current example, we used real data from Wilford et al. (2020) and discussed how to fit these models in R, interpret model parameters, extract predicted probabilities and marginal effects, and visualize the results.

Comparing our analysis with that conducted by Wilford et al. (2020) we demonstrated that using the traditional approach (i.e., *t*-test) to analyze accuracy data can lead to inaccurate inferences. While we did reproduce the results from Wilford et al. (2020), our use of a beta regression model, which accounts for both the mean and precision, revealed no significant effect of fluency. By fitting a zero-inflated beta model, which models the structural 0s in the data, we did find an effect of fluency, but not on the mean component. We found an effect of fluency on the zero-inflated component. Specifically, those in the disfluent condition were more likely to perform more poorly (with a higher probability of 0s). Here, we see fluency does not affect mean accuracy, but instead influences the structural zero part of the model. Fluency might influence specific aspects of performance (such as the likelihood of complete failure) rather than general performance levels. This finding has important theoretical implications and should be included in discussions of fluency effects. Additionally, we demonstrated how to fit data with zeros and ones using the ZOIB and ordered beta models. These analyses Again these analyses highlighted the importance of fitting a model to the data you have.

Overall, this tutorial serves to highlight the importance of modeling the data you have. Modeling your data appropriately can provide deeper and richer insights than those obtained through traditional measures. With this tutorial, researchers now have some tools to analyze their data properly, allowing them to uncover patterns, make accurate predictions, and support their findings with robust statistical evidence. By leveraging these modeling techniques, researchers can enhance the validity and reliability of their studies, leading to more informed decisions and advancements in their respective fields.

References

Arel-Bundock, V. (2024). *Marginaleffects: Predictions, comparisons, slopes, marginal means, and hypothesis tests*. https://CRAN.R-project.org/package=marginaleffects

Bartlett, M. S. (1936). The Square Root Transformation in Analysis of Variance. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *3*(1), 68–78. https://doi.org/10.2307/2983678

Bendixen, T., & Purzycki, B. G. (2023). Cognitive and cultural models in psychological science: A tutorial on modeling free-list data as a dependent variable in Bayesian regression. *Psychological Methods*. https://doi.org/10.1037/met0000553

Brooks, M. E., Kristensen, K., van, K. J., Magnusson, A., Berg, C. W., Nielsen, A., Skaug, H. J., Maechler, M., & Bolker, B. M. (2017). *glmmTMB balances speed and flexibility among packages for zero-inflated generalized linear mixed modeling*. *9*. https://doi.org/10.32614/RJ-2017-066

Bürkner, P.-C. (2017). *Brms: An r package for bayesian multilevel models using stan*. *80*. https://doi.org/10.18637/jss.v080.i01

Bürkner, P.-C., & Vuorre, M. (2019). Ordinal Regression Models in Psychology: A Tutorial. *Advances in Methods and Practices in Psychological Science*, *2*(1), 77–101. https://doi.org/10.1177/2515245918823199

Cribari-Neto, F., & Zeileis, A. (2010). *Beta regression in r*. *34*. https://doi.org/10.18637/jss.v034.i02

Ferrari, S., & Cribari-Neto, F. (2004). Beta Regression for Modelling Rates and Proportions. *Journal of Applied Statistics*, *31*(7), 799–815. https://doi.org/10.1080/0266476042000214501

Fullerton, A. S., & Anderson, K. F. (2021). Ordered Regression Models: a Tutorial. *Prevention Science*, *24*(3), 431–443. https://doi.org/10.1007/s11121-021-01302-y

Gabry, J., Češnovar, R., Johnson, A., & Bronder, S. (2024). *Cmdstanr: R interface to 'CmdStan'*. https://mc-stan.org/cmdstanr/

Heiss, A. (2021). *A guide to modeling proportions with bayesian beta and zero-inflated beta regression models*. http://dx.doi.org/10.59350/7p1a4-0tw75

Kubinec, R. (2022). Ordered Beta Regression: A Parsimonious, Well-Fitting Model for Continuous Data with Lower and Upper Bounds. *Political Analysis*, *31*(4), 519–536. https://doi.org/10.1017/pan.2022.20

Kubinec, R. (2023). *Ordbetareg: Ordered beta regression models with 'brms'*. https://CRAN.R-project.org/package=ordbetareg

Lüdecke, D. (2018). *Ggeffects: Tidy data frames of marginal effects from regression models*. *3*, 772. https://doi.org/10.21105/joss.00772

Lüdecke, D., Ben-Shachar, M. S., Patil, I., Wiernik, B. M., Bacher, E., Thériault, R., & Makowski, D. (2022). *Easystats: Framework for easy statistical modeling, visualization, and reporting*. https://easystats.github.io/easystats/

Makowski, D., Ben-Shachar, M. S., Chen, S. H. A., & Lüdecke, D. (2019). Indices of effect existence and significance in the bayesian framework. *Frontiers in Psychology*, *10*. https://doi.org/10.3389/fpsyg.2019.02767

Makowski, D., Ben-Shachar, M., & Lüdecke, D. (2019). bayestestR: Describing effects and their uncertainty, existence and significance within the bayesian framework. *Journal of Open Source Software*, *4*(40), 1541. https://doi.org/10.21105/joss.01541

Paolino, P. (2001). Maximum Likelihood Estimation of Models with Beta-Distributed Dependent Variables. *Political Analysis*, *9*(4), 325–346. https://doi.org/10.1093/oxfordjournals.pan.a004873

R Core Team. (2024). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Vuorre, M. (2019, February 18). *How to Analyze Visual Analog (Slider) Scale Data?* https://vuorre.com/posts/2019-02-18-analyze-analog-scale-ratings-with-zero-one-inflated-beta-models

Wilford, M. M., Kurpad, N., Platt, M., & Weinstein-Jones, Y. (2020). Lecturer fluency can impact students' judgments of learning and actual learning performance. *Applied Cognitive Psychology*, *34*(6), 1444–1456. https://doi.org/10.1002/acp.3724

Appendix

Title for Appendix